Improvements to Lattice Drawing With fca.sty

Tobias Schlemmer

Dresden, Germany

Abstract

For documenting theoretical and empirical results with of Formal Concept Analysis Bernhard Ganter provided a LaTeX package that allows to typeset Formal Contexts and Line diagrams of Lattices and ordered sets. This package has been heavily reworked during the last years. Here a short status of the achievements and open challenges shall be given.

Keywords

LaTeX package, typesetting FCA, typesetting, formal context, lattice diagrams

1. Introduction

Bernhard Ganter's `fca.sty` is a LaTeX [1] package for typesetting Formal Concept Analysis [2]. This includes special symbols, Formal Contexts, and Lattice Diagrams. This package has been overhauled by the author in the last two years [3]. In the result, the package has improved support for formal contexts and drawing line diagrams.

The intention behind this effort was no less than to improve the typographical quality of papers about Formal Concept Analysis and related subjects. Furthermore, the package should provide a maximum amount of compatibility to existing LaTeX code based on former versions of `fca.sty`.

The main changes are:

- remove limits to the number of columns, rows, concepts, etc. of formal contexts and concept lattices,
- add a parser for Burmeister Context files,
- add an interface to allow a arbitrary LaTeX code for symbols in context tables – this also includes new symbols and colouring of crosses,
- use PGF [4] as backend for simple lattice diagrams and Ti*k*Z [4] for more sophisticated documentations
- expose the improvements from these packages to the users.

In the following sections these changes are shortly introduced one by one. These improvements offer tools that can help to improve the quality of publications about Formal Concept Analysis, however they do not reach this goal. Several hurdles lie on its way. Some of these shall be discussed at the end of this paper.

2. Keeping things separated

The new `fca.sty` consequently uses prefixes to macros and environments in order to avoid interference with other packages. In most cases this should be transparent to the users. However, this cannot be fully avoided:

FCA4AI 2024: The 12th International Workshop "What can FCA do for Artificial Intelligence?, October 19 2024, Santiago de Compostela, Spain

Tobias.Schlemmer@web.de (T. Schlemmer)
https://schlemmersoft.de/ (T. Schlemmer)
0000-0003-4350-3110 (T. Schlemmer)

- Global configuration macros that originally didn't start with `\fca` or `cxt` must be adapted to the new system. Inside the `cxt` and `diagram` environments these macros are provided without prefix in order to avoid unnecessary errors.
- All new diagram styles (see below) must be called using their full path starting with `/fca/` when accessed from outside of FCA macros. If possible they are mapped to the corresponding `/pgf/` or `/tikz/` styles. Details are given in the documentation of `fca.sty`.

3. Improvements to formal contexts

Formula 1	1.	2.	disqualified
Verstappen	×		
Hamilton		×	
Leclerc		×	×

```
\begin{cxt}
   \cxtinput{formula1.cxt}
\end{cxt}
```

Figure 1: A formal context loaded from a Burmeister file. Left: Context, Right: source code.

Support for typesetting formal contexts had been added to `fca.sty` long ago. This support had its limitations. Some of them have been lifted. The way how object and attribute names are stored has been reworked as well as the table header generation. So the number of possible columns is not limited by the package, anymore. The general TEXnical limits should be large enough, even for unusual useage of the package: the number of possible macros, the maximum counter value, and the available memory.

New macros have been introduced that allow the definition of new symbols, and redefine existing ones. The characters denoting the symbols are stored as macros. So they can be defined to consume arguments. Additionally, digits have been predefined so that simple many-valued contexts can be typeset without additional setup. In cases where it is really necessary, it is a new macro allows to inject arbitrary code in the `tabular` environment of the context.

An optional positional argument has been added to the `cxt` environment, so that it is easier to place them in multi column environments. These improvements are demonstrated in Fig. 2.

Last but not least, a parser for contexts in the Burmeister format has been integrated into the packageDanke, as shown in Fig. 1. This parser maps the different parts of a `.cxt` file to the corresponding macros of a `cxt` environment. So markup or special signs can be typeset in the same way as in the corresponding macros of the LATEX environment `cxt`.

4. Lattice diagrams

Lots of work has been invested into the rejuvenation of the diagram code. The syntax has been carefully adapted such that existing diagrams can be directly integrated in new documents, or they need only minor adjustments. An example is given in Fig. 3. Each vertex has a name (traditionally a number) and coordinates. The edges and labels are anchored using these names. Additionally, labels have a printed description and can be shifted relatively to their position.

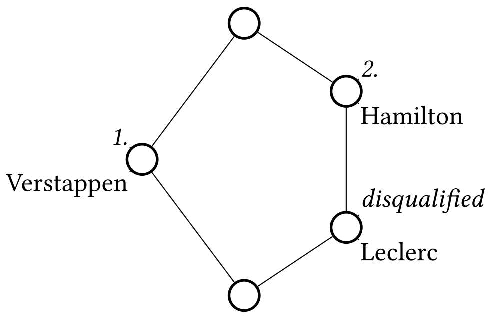
In order to allow a consistent appearance in sophisticated diagrams, the different elements of a diagram are organised in layers.

The original `fca.sty` package used the `\emlines` macro from the emTEX distribution in combination with standard LATEXs `picture` environment. Unfortunately, support for the emTEX specials has been removed from current TEX distributions, so the lines disappear from the diagrams. The package `tsemlines` [5] depends on Ti*k*Z. So it is more a quick hack than a lightweight solution to this problem. Another goal was to bridge the gap between the very simple `picture` environment and modern graphics

A demo context	1.	2.	disqualified
Verstappen	1		∨
Hamilton	∧	×	⤢
Leclerc	i	23	×
nothing			

```
\begin{cxt}[c]
  \renewcommand{\fcaCxtArrowStyle}{\footnotesize\color{red
      }}%
  \fcaNewContextChar{v}{\cxtrlap{$\vee$}}
  \fcaNewContextChar{n}[1]{\cxtrlap{#1}}
  \fcaProvideContextChar{\wedge}{\cxtrlap{$\wedge$}}
  \fcaProvideContextChar{d}{ -- ignored -- }
  \fcaRenewContextChar{d}{\cxtrlap{$i$}}
  \cxtName{A demo context}
  \att{1.}
  \att{2.}
  \atr{disqualified}
  \obj{1.v}{Verstappen}
  \obj{\wedge xb}{Hamilton}
  \obj{dn{23}x}{Leclerc}
  \freeobj{\multicolumn{3}{c|}{}}{nothing}
\end{cxt}
```

Figure 2: A formal context typeset with the new features of the FCA packages. Left: Context, Right: corresponding source code.



```
\setlength{\unitlength}{0.9mm}
\begin{diagram}
  \Node(1)(20,10)
  \Node(2)(35,20)
  \Node(3)(5,30)
  \Node(4)(35,40)
  \Node(5)(20,50)
  %
  \Edge(1)(2)
  \Edge(1)(3)
  \Edge(2)(4)
  \Edge(3)(5)
  \Edge(4)(5)
  %
  \leftAttbox(3){1.}
  \rightAttbox(2){disqualified}
  \rightAttbox(4){2.}
  \leftObjbox(3){Verstappen}
  \rightObjbox(2){Leclerc}
  \rightObjbox(4){Hamilton}
\end{diagram}
```

Figure 3: A diagram example with 5 Vertices and 5 edges, attribute and object labels.

drawing tools like Ti*k*Z. Thus the `diagram` environment is now based on PGF, a new environment `tikzdiagram` has been introduced as a Ti*k*Z version of `diagram`, and the important macros have been enhanced to use the syntax of Ti*k*Z.

Between full Ti*k*Z support and the `picture` like environment there are several intermediate steps. At first the package can be loaded using `\usepackage{fca}`. This uses only the graphics layer of PGF and omits the syntax layer of Ti*k*Z. At this stage a limited support for Ti*k*Z like styles and attributes has been implemented. Naturally this is linked to the styles that have been implemented in `fca.sty`.

On the other end it is possible to load the package using the Ti*k*Z macro `\usetikzlibrary{fca}`. This enables to use of `diagram` inside a `tikzpicture` environment and the `tikzdiagram` environment which combines both in one environment. Using this approach all drawing macros are mapped to the corresponding Ti*k*Z macros which enables full Ti*k*Z support.

It is also possible to use `\usepackage{fca}` after loading Ti*k*Z. Both approaches enable additional styles to be used in a `diagram` environment, as `fca.sty` uses similar internals to Ti*k*Z. However, future versions of Ti*k*Z may cause errors and the set of supported styles may change depending on the Ti*k*Z version. So the latter approach is not recommended.

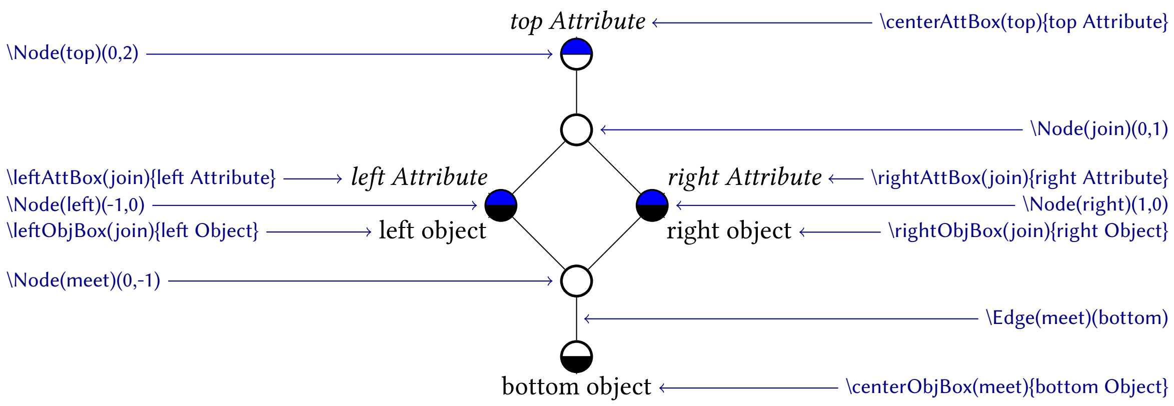
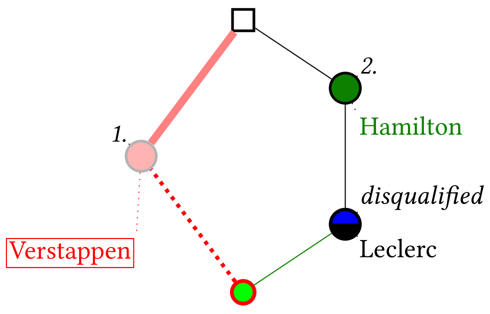

top Attribute
\centerAttBox(top){top Attribute}
\Node(top)(0,2)
\Node(join)(0,1)
\leftAttBox(join){left Attribute}
left Attribute
right Attribute
\rightAttBox(join){right Attribute}
\Node(left)(-1,0)
\Node(right)(1,0)
\leftObjBox(join){left Object}
left object
right object
\rightObjBox(join){right Object}
\Node(meet)(0,-1)
\Edge(meet)(bottom)
bottom object
\centerObjBox(meet){bottom Object}

Figure 4: Elements of a `diagram` environment. The diagram shows a concept lattice. Arrows indicate which code is used to draw certain elements. The diagram is drawn using a `tikzdiagram` environment.



```
\setlength{\unitlength}{0.9mm}
\definecolor{darkgreen}{rgb
    }{0.05,0.5,0.}
\begin{diagram}
  \Node[draw=red, fill=green,
        line width=.5mm,
        radius=1.5mm](1)(20,10)
  \Node(2)(35,20)
  \Node[opacity=0.3, fill=red](3)(5,30)
  \Node[fill=darkgreen](4)(35,40)
  \Node[/tikz/rectangle](5)(20,50)
  %
  {\color{darkgreen}\Edge(1)(2)}
  \Edge[draw=red, dotted,
        line width=1.5pt](1)(3)
  \Edge(2)(4)
  \Edge[draw=red!50,
        line width=1mm](3)(5)
  \Edge(4)(5)
  %
  \rightAttbox[conexp style](2){
       disqualified}
  \leftAttbox(3){1.}
  \rightAttbox(4){2.}
  \leftObjbox[draw=red,inner sep=1pt](3)
       (1,-10){Verstappen}
  \rightObjbox[conexp style](2){Leclerc}
  \rightObjbox[text=darkgreen](4)(0,-2)
       {Hamilton}
\end{diagram}
```

Figure 5: An overly styled concept lattice demonstrating different kinds of markup including the use of Ti*k*Z styles.

Unfortunately, Ti*k*Z and the `picture` environment have different base units. While `picture` only allows to define one unit length, PGF and Ti*k*Z apply at least two affine transformations from the input to the output file. Though the behaviour of a `picture` environment can be emulated in PGF/Ti*k*Z, this behaviour is unstable and counter-intuitive to new users. The compatibility issue is solved using the following compromise:

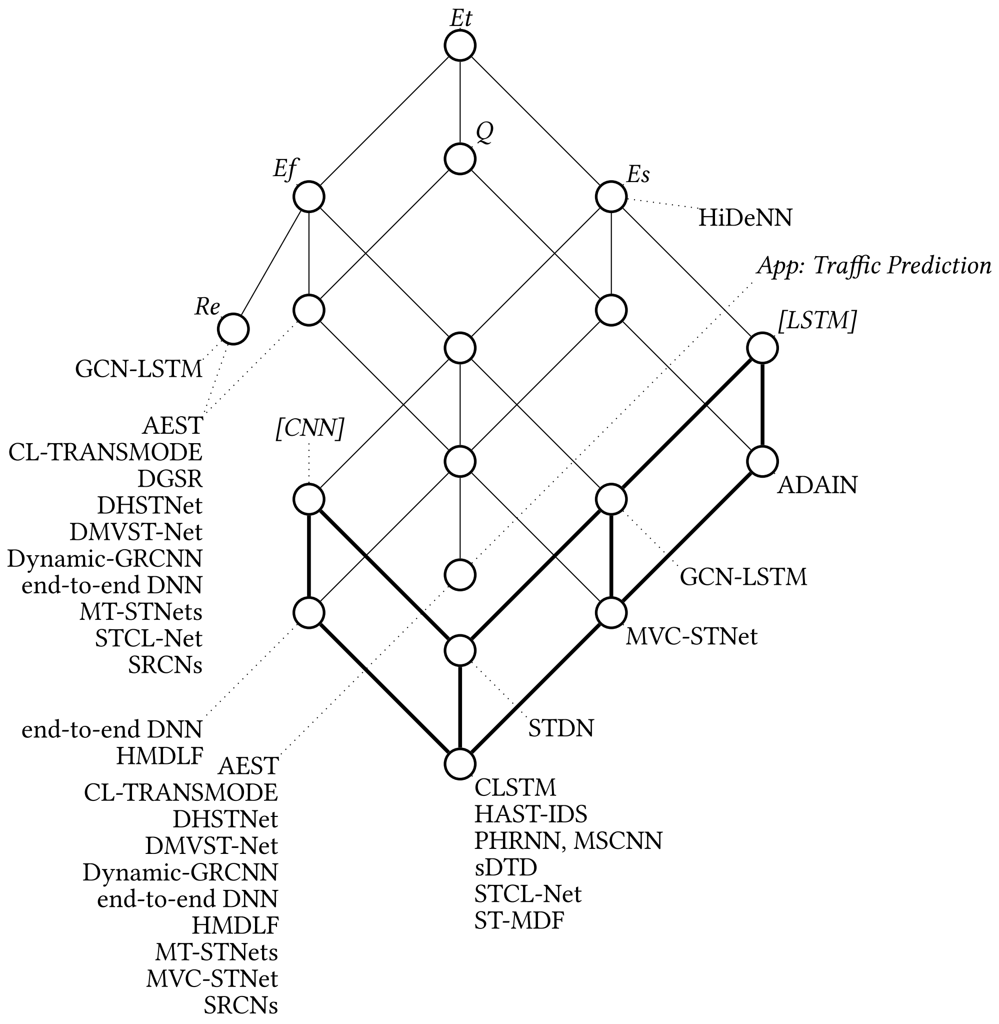


Figure 6: An iceberg lattice [6, Fig. 8], reworked for readability according to the standards defined in [2] with emphasis on the lower part.

- the `digaram` environment uses the old coordinate system if it is located outside of any graphics environment or inside a `pgfpicture` environment,
- the `tikzdiagram` environment and `diagram` inside `tikzpicture` use the tikz coordinate system.
- translation of old diagrams into TikZ diagrams can be easily done by opening the diagram with `\begin{tikzdiagram}[x=\unitlength,y=\unitlengh,...]`.
- All configuration macros in `diagram` environments are either available directly in `tikzdiagram` or can be expressed using style options.

As a real-world example, in Fig. 6 an iceberg lattice [6, Fig. 8] has been reworked using a chain decomposition layout exploiting the `calc` library of TikZ and the style used in [2]. As usual in Formal Concept Analysis, tags for attributes are drawn only on the highest concept node of their occurrence and apply to all concepts that can be connected with only rising lines to the corresponding label. This is in principle also true with inverse direction for object labels, which are valid for every node that can be connected going strictly upwards following the lines. However, many of the objects are attached to nodes that are not visible in the diagram. Their names are repeated at the lowest visible node, which often leads to multiple occurrences of he same name in different labels. Where appropriate, labels are reused for multiple nodes.

The lower part of the diagram is drawn with bolder lines. This part is referenced in an emphasised discussion in the text of the given article.

5. Open issues

It is planned to publish the package on CTAN, so that it can be integrated in the standard LaTeX distributions and Docker images. And despite the fact that the package is perfectly usable, some issues arise. However, it currently does not fulfil its primary goal: to improve the typographical quality of published diagrams in formal concept analysis. As it can be seen, in this paper both high-quality as well as low-quality typesetting is possible with this package. So how can it be modified to achieve this goal?

One approach would be to allow only good diagrams or make it at least hard to draw bad diagrams. One could argue that LaTeX also makes it hard to change dangerous parameters. This is impossible on the technical level as no algorithm exists that can check whether a diagram is good or bad. This decision depends on the writer's intention. Formalising this intention is nearly impossible. The levers we can pull (or not) are basically features and documentation. If we remove all possibly dangerous features from the package or its documentation, the package would be very inflexible. And it would be nearly impossible to interact with other graphical content.

On the other hand, documenting the package as a reference that simply lists all features, and encouraging people to play around with these features, also leads to over-styled diagrams that are hard to understand (cf. Fig. 2 and Fig. 5). Recall: Typography does not consider the taste of the author but studies how to format things such that they can be easily understood by the readers.

Also here, LaTeX can be used as a reference. Since the first version of LaTeX more and more features got configurable. So at first it simplified the use of TeX and then, it simplified the change of the layout.

This is the main issue that blocks publication on CTAN. Currently it is unclear, how to solve it. It seems as if a compromise could be to proper organise the information for the documentation. Other packages like Ti*k*Z and the `beamer` start with tutorials on their subjects. The difficulty of this approach lies in the fact, that such a tutorial should satisfy all stakeholders.

Other issues contain small inconsistencies between the PGF and the Ti*k*Z implementation of the diagram drawing code. These will be ironed out with the growth of the reference section in the documentation. However also the development of the reference is influenced by the above issue.

6. Conclusion

Despite the issues the new version of `fca.sty` is a powerful and usable package that provides:

- Certain symbols for Formal Concept Analysis,
- Context tables drawn from LaTeX code and Burmeister context files,
- Lattice Diagrams that can be drawn and enhanced with annotations in PGF and Ti*k*Z environments.

Happy TeXing!

Declaration on Generative AI

The author has not employed any Generative AI tools.

References

[1] L. Lamport, LaTeX (2nd ed.): a document preparation system: user's guide and reference manual, Addison-Wesley Longman Publishing Co., Inc., USA, 1994.

[2] B. Ganter, R. Wille, Formal concept analysis: mathematical foundations, Springer, Berlin, 1999.

[3] B. Ganter, T. Schlemmer, fca.sty, Online Ressource: https://github.com/keinstein/latex-fca, 2024. Download via https://github.com/keinstein/latex-fca/archive/refs/heads/master.zip.

[4] T. Tantau, The TikZ and PGF Packages, 2024. URL: https://github.com/pgf-tikz/pgf/.

[5] T. Schlemmer, tsemlines, 2024. URL: https://ctan.org/pkg/tsemlines.

[6] T. Man, V. Y. Osipov, N. Zhukova, A. Subbotin, D. I. Ignatov, Neural networks for intelligent multilevel control of artificial and natural objects based on data fusion: A survey, Information Fusion 110 (2024) 102427. doi:10.1016/j.inffus.2024.102427.